Text Analysis

from google.colab import files

uploaded = files.upload()

import pandas as pd

import io

df = pd.read\_csv(io.BytesIO(uploaded['Input.xlsx - Sheet1.csv']))

import requests

import bs4 as bfs

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from textblob import TextBlob

import numpy as np

import re

# Install openpyxl if not already installed

!pip install openpyxl

nltk.download('stopwords')

nltk.download('punkt')

To see URLs from the file:

li = [url for url in df['URL']]

li

# Fetch and parse articles

text = [bfs.BeautifulSoup(requests.get(url, headers={"User-Agent": "XY"}).content, 'html.parser') for url in li]

# Extract text content from articles

articles = []

for t in text:

    post\_content = t.find(attrs={"class": "td-post-content"})

    if post\_content:

        articles.append(post\_content.text.replace('\n', ''))

    else:

        articles.append("No content found for this article")

# Now, the 'articles' list contains the extracted text or a placeholder message if the element is not found.

# Tokenize and clean text

stop\_words = set(stopwords.words('english'))

cleaned\_articles = [' '.join(re.sub(r'[^\w\s]', '', word) for word in article.split() if word.lower() not in stop\_words) for article in articles]

#Import additional library and packages:

from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download('vader\_lexicon')

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('vader\_lexicon')

# Sentiment Analysis

sid = SentimentIntensityAnalyzer()

df['SENTIMENT SCORE'] = [sid.polarity\_scores(article)['compound'] for article in cleaned\_articles]

# TextBlob Subjectivity

df['SUBJECTIVITY'] = [TextBlob(article).sentiment.subjectivity for article in cleaned\_articles]

#Upload dictionary for sentiment analysis

from google.colab import files

uploaded = files.upload()

import io

dictionary = pd.read\_csv(io.BytesIO(uploaded['Loughran-McDonald\_MasterDictionary\_1993-2021.csv']))

positive\_words = list(dictionary[dictionary['Positive'] == 2009]['Word'])

The following code can be excluded:

positive\_score = [0] \* len(articles)

for i in range(len(articles)):

    for word in positive\_words:

        for letter in cleaned\_articles[i].lower().split(' '):

            if letter == word:

                positive\_score[i] += 1

#Calculate the “POSITIVE” and “NEGATIVE” words and scores

negative\_words = set(dictionary[dictionary['Negative'] == 2009]['Word'])

negative\_score = [

    sum(1 for word in cleaned\_article.upper().split() if word in negative\_words)

    for cleaned\_article in cleaned\_articles

]

positive\_words = set(dictionary[dictionary['Positive'] == 2009]['Word'])

positive\_score = [

    sum(1 for word in cleaned\_article.upper().split() if word in positive\_words)

    for cleaned\_article in cleaned\_articles

]

df['POSITIVE SCORE'] = positive\_score

df['NEGATIVE SCORE'] = negative\_score

#Calculate “WORD COUNT” and “POSITIVE WORD COUNT”

# Assuming 'cleaned\_articles' is a list of cleaned article texts

df['WORD COUNT'] = [len(word\_tokenize(article)) for article in cleaned\_articles]

# Assuming 'cleaned\_articles' is a list of cleaned article texts

# Function to check if a word is complex

def is\_complex(word):

    return len([char for char in word if char.lower() in 'aeiou']) > 2

df['COMPLEX WORD COUNT'] = [

    sum(1 for word in word\_tokenize(article) if is\_complex(word))

    for article in cleaned\_articles]

total\_characters = [len(article.replace(" ", "")) for article in cleaned\_articles]

word\_count = [len(word\_tokenize(article)) for article in cleaned\_articles]

#Calculate “WORD LENGTH” and “POSITIVE WORD LENGTH”

# Assuming 'total\_characters' is a list of total characters in each article

# and 'word\_count' is a list of word counts for each article

df['AVG WORD LENGTH'] = np.array(total\_characters) / np.array(word\_count)

df

complex\_word\_count = [

    sum(1 for word in word\_tokenize(article) if is\_complex(word))

    for article in cleaned\_articles

]

word\_count = [len(word\_tokenize(article)) for article in cleaned\_articles]

#Calculate the “PERCENTAGE of COMPLEX WORDS”

# Assuming 'complex\_word\_count' is a list of complex word counts for each article

# and 'word\_count' is a list of word counts for each article

df['PERCENTAGE OF COMPLEX WORDS'] = np.array(complex\_word\_count) / np.array(word\_count)

df